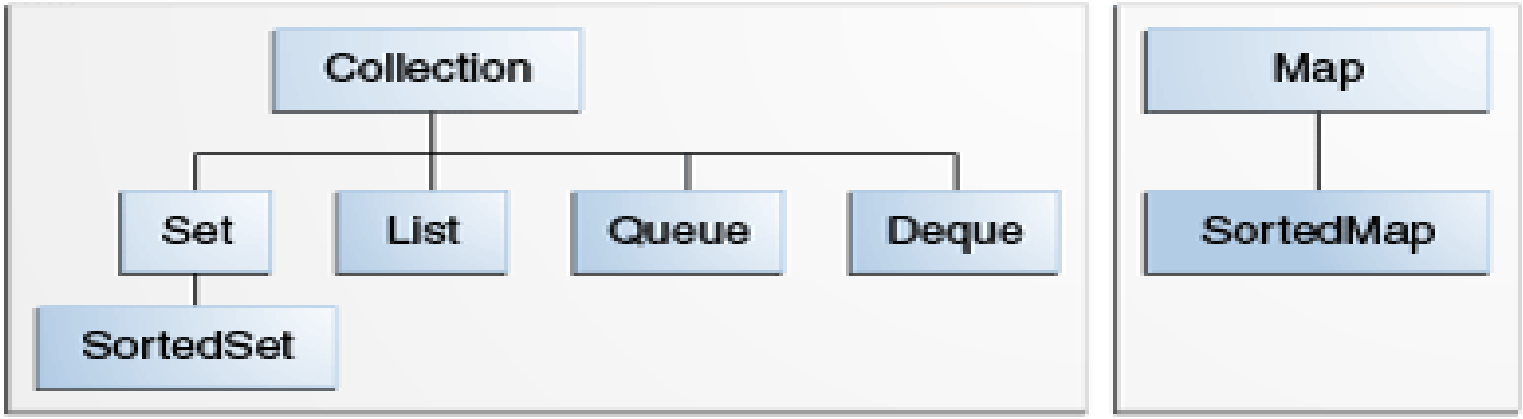
Java Collection Framework & Stream API

IF2210 – Semester II 2020/2021

Java Collection Framework

- A *collections framework* is a unified architecture for representing and manipulating collections.
- All collections frameworks contain the following:
  - **Interfaces:** the abstract data types that represent collections
  - **Implementations:** the concrete implementations of the collection interfaces
  - **Algorithms:** the methods that perform useful computations, such as searching and sorting, on objects that implement collection interfaces

Core Collection Interfaces



- All the core collection interfaces are generic, contoh:

```
public interface Collection<E>...
```

- Dokumentasi lengkap baca: https://docs.oracle.com/javase/tutorial/collections/index.html

Collection Interface

- A `Collection` represents a group of objects known as its elements.
- Used to pass around collections of objects where maximum generality is desired.

Collection Interface declares...

- Methods that perform basic operations, e.g.:
  `int size()`, `boolean isEmpty()`,
  `boolean contains(Object element)`, `boolean add(E element)`,
  `boolean remove(Object element)`, and `Iterator<E> iterator()`.
- Methods that operate on entire collections, e.g.:
  `boolean containsAll(Collection<?> c)`,
  `boolean addAll(Collection<? extends E> c)`,
  `boolean removeAll(Collection<?> c)`,
  `boolean retainAll(Collection<?> c)`, and `void clear()`.
- Additional methods for array operations e.g.: `Object[] toArray()` and `<T> T[] toArray(T[] a)`.

Set Interface

- A `Set` is a `Collection` that cannot contain duplicate elements.
- Contains only methods inherited from `Collection` and adds the restriction that duplicate elements are prohibited.
- Three general-purpose `Set` implementations: `HashSet`, `TreeSet`, and `LinkedHashSet`.

List Interface

- A `List` is a `Collection` that preserves insertion order.
- May contain duplicate elements.
- includes operations for the following:
  - Positional access — manipulates elements based on their numerical position in the list. This includes methods such as `get`, `set`, `add`, `addAll`, and `remove`.
  - Search — searches for a specified object in the list and returns its numerical position. Search methods include `indexOf` and `lastIndexOf`.
  - Iteration — extends Iterator semantics to take advantage of the list's sequential nature. The `listIterator` methods provide this behavior.
  - Range-view — The `sublist` method performs arbitrary range operations on the list.
- Two general-purpose List implementations: `ArrayList` and `LinkedList`

Queue Interface

- A `Queue` is a collection for holding elements prior to processing.
- Besides basic `Collection` operations, queues provide additional insertion, removal, and inspection operations.
- Queues typically, but not necessarily, order elements in a FIFO (first-in-first-out) manner.

Deque Interface

- Usually pronounced as *deck*, a *deque* is a double-ended-queue.
- A double-ended-queue is a linear collection of elements that supports the insertion and removal of elements at both end points.
- Methods are provided to insert, remove, and examine the elements.
- Predefined classes like `ArrayDeque` and `LinkedList` implement the `Deque` interface.

Map Interface

- A `Map` is an object that maps keys to values.
- A map cannot contain duplicate keys: Each key can map to at most one value.
- Includes methods for basic operations (such as `put`, `get`, `remove`, `containsKey`, `containsValue`, `size`, and `empty`), bulk operations (such as `putAll` and `clear`), and collection views (such as `keySet`, `entrySet`, and `values`).
- Three general-purpose `Map` implementations: `HashMap`, `TreeMap`, and `LinkedHashMap`.

Contoh Penggunaan

- `ArrayList` adalah salah satu implementasi dari interface `List`.

```
List<Person> myList = new ArrayList<Person>();
myList.add( new Person("amir") );
Person p = myList.get(0);

ArrayList<String> ar = new ArrayList<String>();
ar.add("satu");
```

Stream API (1)

- Sebelum Java 8, proses yang melibatkan elemen-elemen `Collection` harus menggunakan *loop construct*.
- Sejak Java 8 ada Stream API dan Lambda expression yang memungkinkan proses dilakukan secara *fluent*.
  - Contoh: mencetak isi list hanya yang dimulai dengan huruf "c", dijadikan uppercase, terurut secara alfabetis

```
List<String> myList =
    Arrays.asList("a1", "a2", "b1", "c2", "c1");
```

```java
List<String> myList =
    Arrays.asList("a1", "a2", "b1", "c2", "c1");

// dengan loop:
List<String> target = new ArrayList<>();
for (String s: myList) {
    if (s.startsWith("c")) {
        target.add(s.toUpperCase());
    }
}
Collections.sort(target);
for (String s: target) {
    System.out.println(s);
}

// dengan Stream API:
myList
    .stream()
    .filter(s -> s.startsWith("c"))
    .map(String::toUpperCase)
    .sorted()
    .forEach(System.out::println);
```

Stream API (2)

- Ingat kembali konsep collection di OOP.
- Pada Java Stream API secara umum ada 4 jenis proses:
  - `filter`: dari *stream* menjadi *stream* bertipe sama dengan jumlah elemen yang bisa lebih sedikit dari *stream* aslinya.
  - `map`: dari *stream* menjadi *stream* (bisa bertipe lain) dengan jumlah elemen yang sama dengan *stream* aslinya.
  - `reduce`: dari *stream* menjadi satu nilai.
  - `forEach`: melakukan sesuatu terhadap semua elemen *stream*.
- Baca: https://winterbe.com/posts/2014/07/31/java8-stream-tutorial-examples/